4. B r u d a r u, O. — *A Note on the Design of Linear Systolic Arrays for Some Real Functions Computation*. Bulletin of the Polytechnical Institute of Iaşi (to appear).
5. B r u d a r u, O. — *Systolic Computation of Some Real Functions Given by Acyclic Digraphs*, 33. Internationalen Wissenschäftlichen Kolloquium der Technische Hochschule Illmenau 1988, Vortragsreiche „Graphen und Nettwerke-Theorie und Anwendungen", pp. 167—170.
6. I l i o i, C. — *Numerical-Analysis Approximation Techniques*, Univ. „Al. I. Cuza" Iaşi, Department of Mathematics, 1983 (in Romanian).
7. Q u i n t o n, P. — *An Introduction to Systolic Computation*, Further Parallel Computer, LNCS 272, Springer Verlag, 1986, pp. 387—400.
8. R a o S. K., K a i l a t h, I. — *Architecture Design for Regular Iterative Algorithms*, Systolic Signal Processing Systems, (E. E. Swartzlander ed.), Marcel Dekker Inc., New York-Basel, 1987.
9. S c h e i d, F. — *Theory and Problems of Numerical Analysis*, Schaum's Outline Series, McGraw-Hill, New York, 1968.

Received 28 III. 1989

Polytechnic Institute of Iaşi
Computing Centre
Splai Bahlui 63, 6600—Iaşi, Romania

An. şt. Univ. „Al. I. Cuza" Iaşi
Informatică, 35 (1989)

FROM SYSTOLIC ARRAYS TO NEURAL NETWORKS

BY

VALERIU BEIU

§1. **Introduction.** With the advances made by technology it is clear that parallel architecture are flourishing and that new solutions are being found to improve their already high performances. Talking about parallel architecture we have to mention what has already been said : "*Whenever a computer designer has reached for a level of performance beyond that of his contemporary technology, parallel processing has been his apprentice*" (IEEE Computer Society Tutorial on Parallel Processing, R. H. Kuhn and D.A. Padna, Eds., 1981). The development of technology and the steps taken in neuroscience and computer science in the last few years have led to a new and growing interest in computational models based on NN (connectionist) models. These models link animal brains and behavior with our knowledge about parallel computation and that on artificial intelligence (AI). The idea of looking directly at massively parallel realizations of intelligent activity promises to be fruitful for the study of both natural and artificial computation. The effort has been directed toward the biological implications but there are also important relations computational theory, hardware and software [H i l l i s 1987, P a l m 1985, P a r k e r 1978].

In comparison with a computer, the brain process the information quite differently. The human memory is distributed, and the notion of program is of no use. Our knowledge and power of inference is due to many tiny neurons which are connected one to another : when we learn these interconnections are modified.

Much of the recent work on massively parallel computation has been carried out by physicist [H o p f i e l d 1986] and examines the emergent behavior of large, unstructured collections of computing units.

Adaptation and learning are treated as ways to improve structured networks, not as a replacement for analysis and design [F e l d m a n 1986].

Animal brains derive much of their power from their very large fan-in and fan-out : each neuron can be connected to about 10^4 of other neurons.

§2. **Systolic systems.** Although a SA or systolic system (SS) can be simply viewed as a set of interconnected Moore machines, the exact specification is slightly complicated. In what follows we will quote from [L e i s e r s o n 1982].

"*The structure of a SS $S(n)$ is given by a machine graph $G=(V, E)$ of n vertices interconnected machines where the vertices in V represent the machines and the directed edges in E represent interconnections between machines. The* *machines operate synchronously by means of a common clock, and time in the system is measured as the number of clock cycles. All the machines in V are Moore machines with the exception of one called the host, which can be viewed as a Turing-equivalent machine (it provides input to and receives output from the SS). To the other machines in the system, however, the hostlooks like just another Moore machine, i.e. signals cannot ripple through it. Based on the machine graph, the neighborhood of a machine* $v \in V$ *is the set of machines with which it communicates:*

$$Neigh(v) = \{w \mid (v, w) \in E \text{ or } (w, v) \in E\} \tag{1}$$

For S(n) to be systolic, it is further required that the Moore machines be small in the following sense. There must exist constants c1, c2, c3 and c4 such that for all n and all $v \in V / \{host\}$:

$|Q_V| \leqslant c1$ *the number of states of a machine is bounded,*
$|T_V| \leqslant c2$ *the number of input symbols for a machine is bounded,*
$|O_V| \leqslant c3$ *the number of output symbols for a machine is bounded.*
$|Neigh(v)| \leqslant c4$ *the number of neighbors of a machine is bounded (i.e. the graph has bounded degree).*

These "smallness" conditions help guarantee that the systolic model corresponds in performance to a physical implementation. They ensure that as the size of the system grows, the amount of hardware needed to implement a given machine remains the same. If the logic in an individual machine were to grow with the system size, the time required for the logic to settle to a stable value could depend on the size of the system. Thus the measure of time in the model would poorly reflect the actual time required in a real implementation. With the smallness condition, however, the amount of hardware required for each machine remains small as the system size grows, and hence the time needed for a machine to change state is independent of the size of the system.

The smallness condition go a long way towards ensuring that the number of clockcycles is a good measure of time in the systolic model. A problem arises when the time required to propagate a signal between machines becomes longer than the time required for the longest combinational-logic delay through a machine. The period of the clock must be at least as long as the longest propagation delay between machines, which means that the independence of the clock period from the system size will not be realized for systems with long interconnections. Fortunately this effect is relatively unimportant for many integrated circuits technologies as propagation delay is typically much shorter than swiching delay. The degree to which the switching delay dominates propagation delay is a measure of the success of the model.

SAs, which have only nearest neighbor connections, are especially attractive for VLSI because propagation delay is insignificant. The independence of clock period from the delay caused by many stages of combinational logic is invalidated when Mealy machines are allowed in a system. Despite the potential for rippling of signals from input to output, it is often easier to design with Mealy logic because global communication can be expressed more easily. For instance, brodcasting, the most mean of global communications can be implemented by letting a signal ripple throughout the system until it reaches all processors.

A semisystolic system is exactly like a SS except that some of the machines may be Mealy machines. All the other requirements of SS apply to semisystolic

systems, but in addition, the output edges from Mealy machines may not form a cycle in the machine graph. This constraint precludes the problems of unclocked state and oscillation which are associated with feedback. No such requirement is needed for output edges from Moore machines in a semisystolic system (see Figure 1) because a changing input to a Moore machine cannot affect its output.

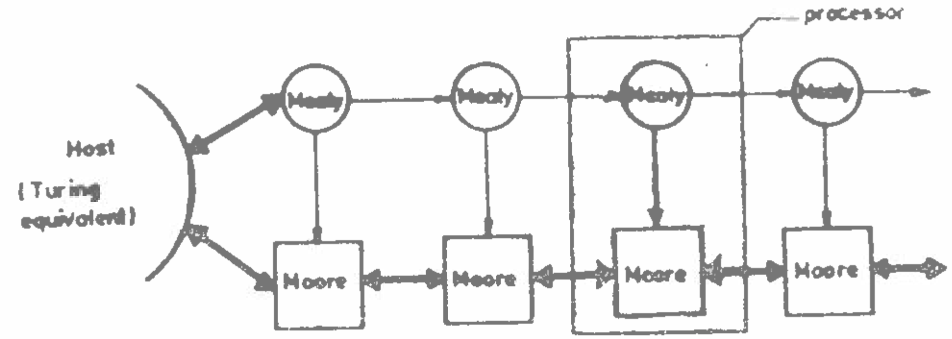


Fig. 1 — Semisystolic system implementing brodcast with Mealy machines and grouping into processors.

Although the machine graph G gives a computational structure for a systolic or semisystolic system, it is often useful to organize groups of machines into processors. Depending upon the logical structure of the system, one grouping might be preferred; for all but trivial systems, several grouping are sure to exist.

Definition. *An undirected bounded-degree graph H is a processor graph for a systolic or semisystolic system S with machine graph G of n vertices if there exists a constant c, independent of n, and a processor mapping θ from the vertices of G (machines) to the vertices of H (processors) such that*

if $(v, w) \in G$, *then either* $\theta v = \theta w$ *or* $(\theta v, \theta w) \in H$ *and*

if $w \in H$, *then* $|\{v \mid w = \theta v\}| \leqslant c$.

In other words, the processor mapping θ tells which machine compose which processor. Two machines which are connected must either be in the same processor, or be in adjacent processor according to the machine graph. To make sure that the size of a processor is independent of system size, at most c machines in the machine graph may be mapped to the same processor".

3. Neural networks. In this paper neural node is a basic computational unit capable of accepting continuous valued inputs and outputing the value of some nonlinear function applied to the weighted sum of its inputs. If $I = (i_1, i_2, \ldots, i_m)$ is the input vector the output is

$$O = \sum_{k=1}^{m} w_k i_k + \theta \tag{2}$$

A network with two hidden layers is obtained by constructing two layers of neural nodes; the inputs to the first layer are inputs to the network, while the inputs to the second layer of neural nodes are the outputs of the first layer. The outputs of the second layer are combined by a single neural node at a third layer that only computes a linear combination of the second layer outputs without nonlinearity.

It is possible that each neural node be different not only in the weights w_k but also in the nonlinearity function. Such a network merely implements a function from R^m to R^n. The interested reader can consult : *Neural Networks* 1988, L i p p m a n n 1987, E c k m i l e r 1988 for more detailed information. It has been known for a long time that any computer program could be simulated by an appropriate NN [M c C u l l o c h 1965]. In [C y b e n k o 1988] a theorem that "*continous valued NN with two hidden layers and any fixed continous sigmoidal nonlinearity can approximate any continous function, arbitrarily well on a compact set*" is demonstrated. These results show that NNs can "approximate" any program or hardware implementation if the network is sufficiently large.

§4. **Comparison.** We give a short comparison of SAs and NNs in what concerns their similarities and differences as well as specific problems which can be implemented by SAs or NNs.

● **Similarities :**

● high parallelism ;
● the elementary processors (cells, nodes, machines, perceptrons,...) are very simple
● typically every elementary processor uses the same algorithm (or there are few different types of processor algorithms) ;
● only simultaneous activation (systolic systems) or fixed activation (delay dependent in the case of semisystolic systems) ;
● are special-purpose systems;
● are inexpensive ;
● are fault-tolerant ;
● are very performant ;
● the areas of problems they can solve are (at the time being) quite similar.

Differences :

● SAs exhibit pipe-line parallelism while NNs implement a distributed parallelism
● NNs cells have only one output ;
● NNs cells use a nonlinear filtering function ;
● as dynamic properties the order in which the cells are activated can be fixed, simultaneous or random ; only NNs cells can be randomly activated ;
● NNs are not special-purpose systems if they are sufficiently large ;
● NNs have unique features for learning and adaption (learning can modify the weights as well as the topology of the network) ;
● while SAs have simple regular and fixed connections, NNs present very complicated and modificable topologies ;
● SAs can easily be made fault-tolerant while NNs are intrinsecally faulttolerant having a great degree of robustness. NNs can deal even with unexpected situations and improve their "algorithm" by adapting connection weights in time ;
● SAs are mainly digital while NNs are primarily analog (actually NNs are simulated using digital circuits but in the final stage of an implementation the hardware should be analog) ;
● SAs are implemented with high speed components to lower the clock period ; NNs could be implemented even with slower components ;
● NNs will probably solve more problems if they can be made large enough ; SAs are from this point limited to our ability to "systolize" classical algorithms ;
● at the time being there are alredy NNs solutions for : knowledge representation, inference and expert system behavior which are not addressed by SAs.

Specific problems for SAs :

● dictionary machines (for data base) ;
● algorithms on graphs ;
● matrix operations ;
● signal processing (convolution, correlation filtering, FFT ;
● operations on integers and polynomials (divide, multiply, square, GCD) ;
● vision (pattern matching, filtering, aliasing, contour, thinning) ;
● combinatorial optimization (for example with "statistical cooling") ;
● speech understanding.

Specific problems for NNs :

● knowledge data base for stochastic information (associative memory,...) ;
● pattern recognition ;
● brightness perception ;
● surface interpolation ;
● localizing edges ;
● locomotion (robotic control using self organization) ;
● speech recognition ;
● knowledge representation ;
● inference ;
● expert system (decision making) ;
● combinatorial optimization.

§5. **The priority queue.** As a very poor example for NNs posibilities we have choosen the classical priority queue [L e i s e r s o n 1982] (see Figure 2).

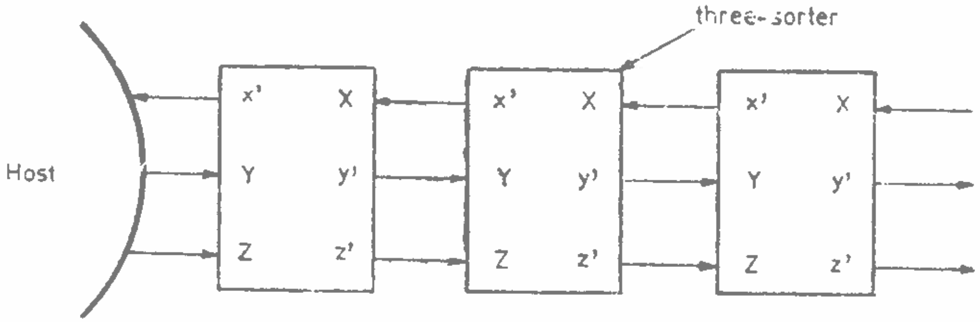


Fig. 2 — A real-time systolic priority queue

A priority queue is a data structure that provides the ability to insert records into a set, and at any time to retrieve from the set the record having the smallest key according to some linear ordering. Thus the structure has to implement INSERT (Q, a) which replaces the set Q with the set $Q \cup \{a\}$, and EXTRACTMIN (Q) which returns the smallest element a of Q and replaces Q with $Q/\{a\}$. Priority queues are usually implemented in software, but a systolic solution is based on identical processors, each capable of sorting three elements and thus called a three-sorter. The three-sorter has three inputs: X, Y, Z and produces three outputs: x', y', z' representing the minimum, median and maximum of the inputs (X, Y, Z).

We are interested in a NN implementation of the three-sorter. The starting idea is to compute the maximum of two numbers using the formula

(3) $$\max(a, b) = \{|a-b| + |a+b|\}/2$$

This leads to the following implementation

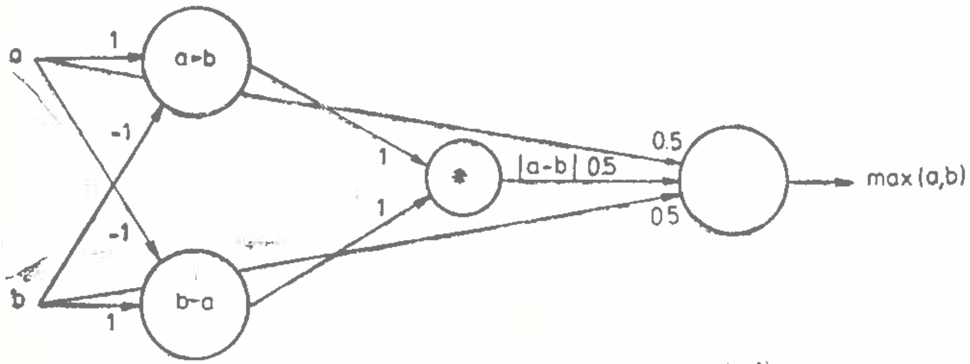


Fig. 3 – The most direct implementation of max (a, b)

It may be seen that the artificial neuron marked with* is futil; so the network becomes:

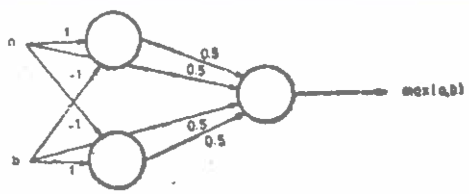


Fig. 4 – Optimized implementation of max (a, b)

The same solution is also shown in [L i p p m a n n 1987]. Using two identical networks we can implement max (a, b, c).

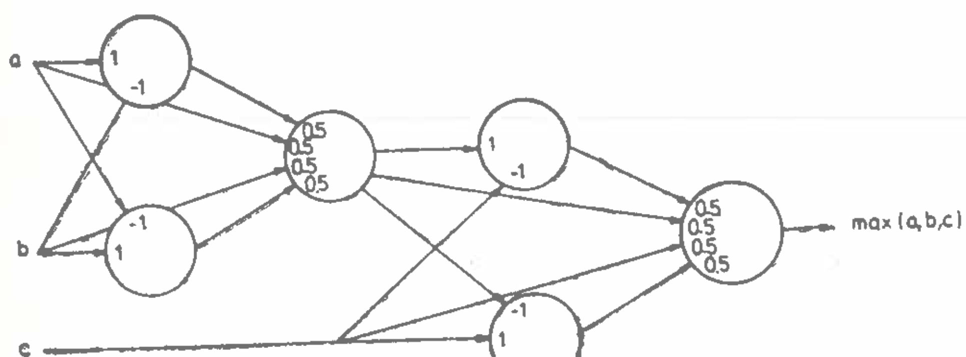


Fig. 5 – Implementing max (a, b, c) with two max (a, b) networks

This solution is certainly correct but it is not optimal in the number of artificial neurones used, nor in the number of stages from input to output (in a NN such optimizations are usually of no importance; in the case being the equivalence with a SA led as to minimize the "delay"). A more simple solution can be derived directly from max (a, b, c)

(4) $$\max(a, b, c) = \max[\max(a, b), c] = \{||a+b| + |a-b| + 2c| + ||a+b| + |a-b| - 2c|\}/4$$

The result is presented in the following figure:

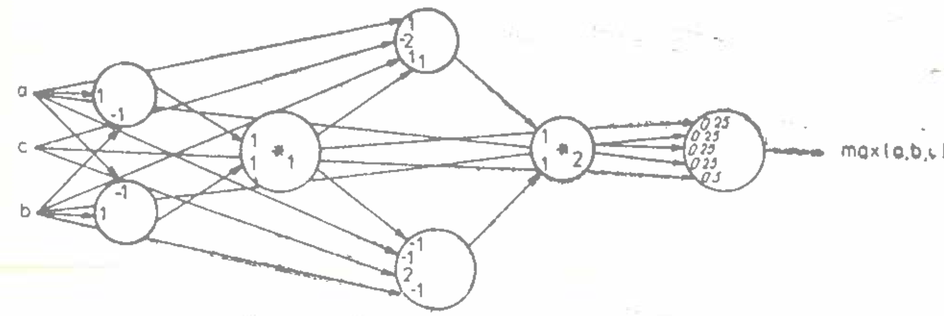


Fig. 6 – "Neural" implementation of equation (4)

Again it may be seen that the artificial neurons marked with * are futil; so the network becomes

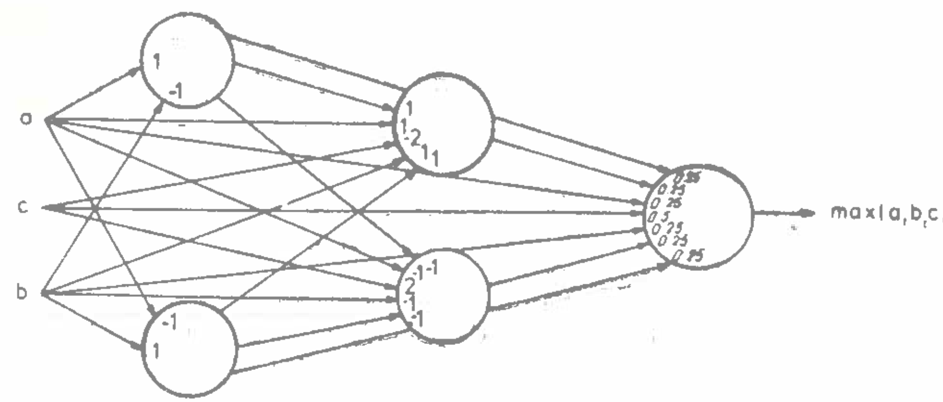


Fig. 7 – Optimal implementation of max (a, b, c)

In similar way we can obtain min (a, b, c) using

(5) $$\min(a, b, c) = \min[\min(a, b), c] = \{||a+b| - |a-b| + 2c| - ||a+b| - |a-b| - 2c|\}/4$$

which is presented in the next figure. For med (a, b, c) we can use

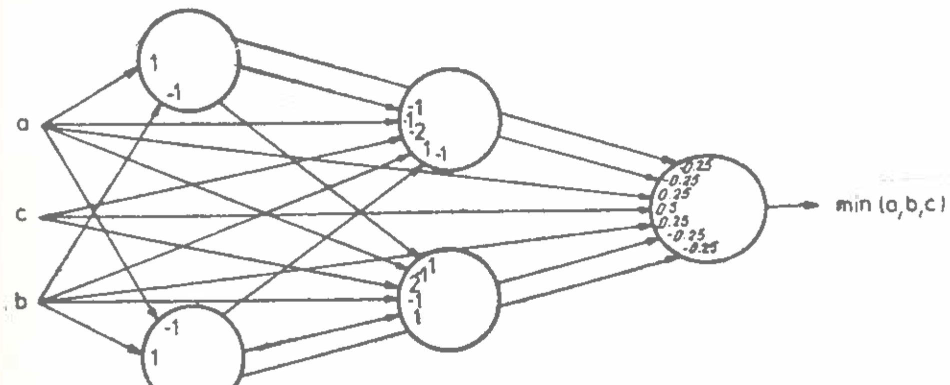


Fig. 8 — The network with modified weights for computing min (a, b, c)

$$\text{med } (a, b, c) = \{a, b, c\} / \{\max (a, b, c), \min (a, b, c)\} \tag{6}$$

Computing the weights now leads, after summations, to

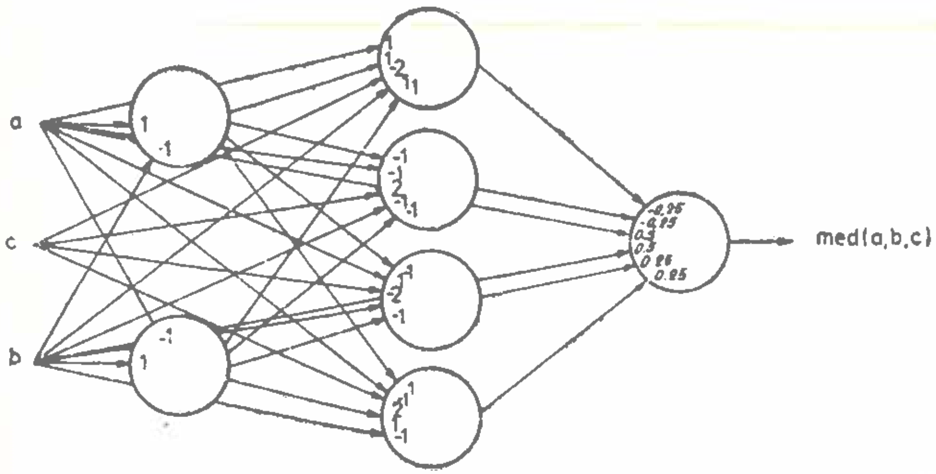


Fig. 9 — **Neural network for med** (a, b, c)

The final implementation of the three-sorter, using artificial neurons is shown in figure 10. It is not a three layer solution, but using "pass neurons" we can derive one (see. Figure 11).

It has be mentioned that this mode of design is not at all specific for NNs: the weights have been computed from formulas while the usual way implies the use of an adaption algorithm and a training sequence to compute these weights. This "reverse engineer" design can only be made for very small portion of a NN, as in our case.

The result is functionally equivalent with a three-sorter: the NN from figure 11 has been simulated with success using NeuroSim [B e i u 1989 a, B e i u 1989 b].

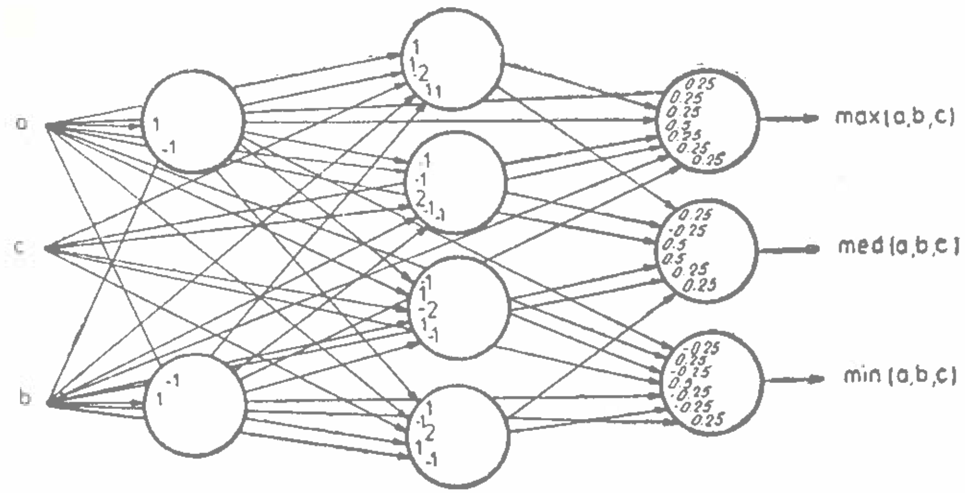


Fig. 10 — A possible three-sorter

6. Conclusions. On a particular example it has been shown that it is possible to reverse engineer a NN. The results have been simulated an proven to be correct. We are currently working at a specific NN implementation of the priority queue which if successful will prove the fact that specific NN solutions are more efficient than the one we have presented.

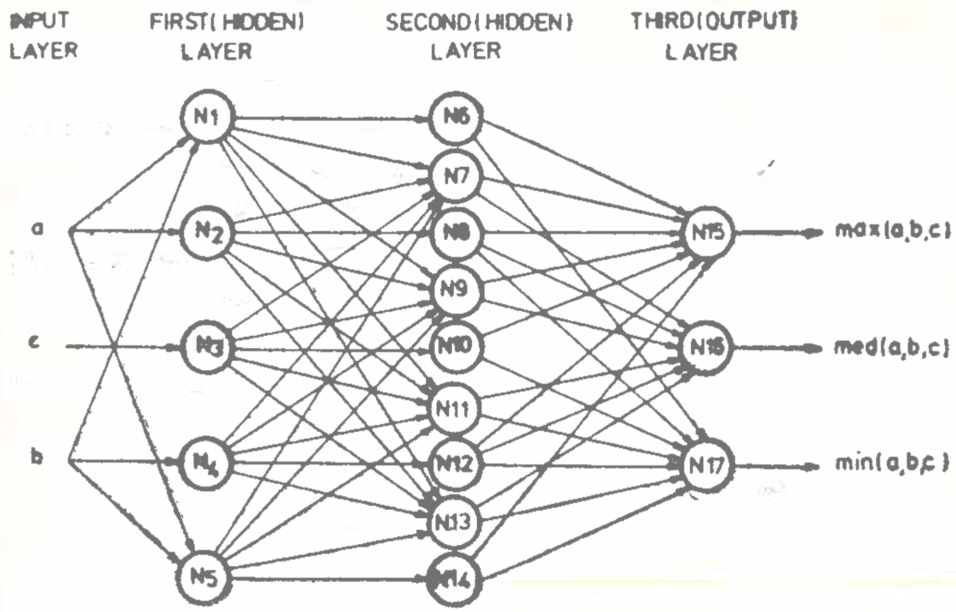
It seems likely that the massively parallel computational solutions evolved by nature will prove useful in computer science and engineering.

Also it is much too early to be certain, it appears that structured NN models could lead to major advances in our ability to automate complex tasks, such those of AI.

Acknowledgements. We would like to express our special thanks to the entire staff of RomTeamXerox-Office Automation Department (INFA) for their help and understanding as this document has been created using *Xerox Ventura Publisher* and a *Xerox 4045 laser printer*.

REFERENCES

1. B e i u, V., F l o r e a, A., G e o r g e s c u, S. — *NeuroSim : A Neural Network Simulator*, accepted for presentation at the 8th CSCS, May 1989 (cancel).
2. B e i u, V., F l o r e a, A., G e o r g e s c u, S. — *The Facilities of the Neuro Sim Environment*, to be published in the Proceedings of the 7th INFO' Iaşi Conference, Oct. 1989.
3. C y b e n k o G., *Continous Valued Neural Networks with Two Hidden Layers are Sufficient* Tech. Rep. TC/CS-88/03, Computer Science Dept., Tufts Univ., Medford, MA, Mar. 1988.
4. E c k m i l l e r, R., M a l s b u r g C. y. v. — *Neural Computers*, Berlin : Springer-Verlag, 1988.
5. F e l d m a n n, J. A. — *Neural Representation of Conceptual Knowledge*, Tech. Rep. 189, Computer Science Dept., Univ., of Rochester, Rochester NY, June 1986.
6. H i l l i s, W. D., B a r n e s, J. — *Programming a Highly Paralel Computer*, Nature 326, 6108, 27—30, Mar. 5 1987.



CONNECTION WEIGHTS

	N1	N2	N3	N4	N5
a	1	1			-1
b				1	
c	-1		1		1

	N6	N7	N8	N9	N10	N11	N12	N13	N14
N1	1	1		-1		-1		1	
N2		1	1	-1		1		-1	
N3		-2		2	1	-2		2	
N4		1		-1		1	1	-1	
N5		1		-1		-1		1	1

	N6	N7	N8	N9	N10	N11	N12	N13	N14
N15	0.25	0.25	0.25	0.25	0.5		0.25		0.25
N16		-0.25	0.5	-0.25		0.25	0.5	0.25	
N17	-0.25		0.25		0.5	-0.25	0.25	-0.25	-0.25

Fig. 11 — Three layer three-sorter derived from figure 10

7. Hopfield, J. J., Tank, D. W. — *Computing with Neural Circuits : A Model*, Science 233, 4761, 625—632, Aug. 8 1986.

8. McCulloch, W. S., Pitts, W. H. — *A Logical Calculus of the Ideas Imminent in Nervous Activity*, Bull. Math Biophys. 5(1943), 115—133 (reprinted : McCulloch W. S., Embodiments Mind, MIT Press, Cambridge, Mass., 1965).

9. Leiseron C. E. — *Area-Efficeint VLSI Computation*, The MIT Press, Cambridge, MA., 1982.

10. Lippmann, R. P. — *An Introduction Computing with Neural Nets*, IEEE ASSP Magazine, 4(2), 4-22, Apr. 1987.

11. Palm, G. — *Associative Networks and Their Information Storage Capacity*, Cognitive Syst, 1, 2, June 1985, 107-118.

12. Parker D. B. — *Learning-logic*, Tech. Rep. 47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, Mass., Apr. 1978.

13. *******. *Neural Networks*, The Official Journal of the International Neural Network Society, 1988.

Received 15.VI.1989

Polytechnic Institute of Bucharest
Department of Computer Science
Spl. Independentei No 313, Sector 6
77206 Bucharest, România